Managing the CERN Batch System with Kubernetes

Luis Fernandez Alvarez[1,*], *Olga* Datskova[1,**], *Ben* Jones[1,***], *Gavin* McCance[1,****]

[1]CERN, 1 Esplanade Des Particules, Geneva, Switzerland

Abstract. The CERN Batch Service faces many challenges in order to get ready for the computing demands of future LHC runs. These challenges require that we look at all potential resources, assessing how efficiently we use them and that we explore different alternatives to exploit opportunistic resources in our infrastructure as well as outside of the CERN computing centre. Several projects, like BEER, Helix Nebula Science Cloud and the new OCRE project, have proven our ability to run batch workloads on a wide range of non-traditional resources. However, the challenge is not only to obtain the raw compute resources needed but how to define an operational model that is cost and time efficient, scalable and flexible enough to adapt to a heterogeneous infrastructure. In order to tackle both the provisioning and operational challenges it was decided to use Kubernetes. By using Kubernetes we benefit from a de-facto standard in containerised environments, available in nearly all cloud providers and surrounded by a vibrant ecosystem of open-source projects. Leveraging Kubernetes' built-in functionality, and other open-source tools such as Helm, Terraform and GitLab CI, we have deployed a first cluster prototype which we discuss in detail. The effort has simplified many of the existing operational procedures we currently have, but has also made us rethink established procedures and assumptions that were only valid in a VM-based cloud environment. This contribution presents how we have adopted Kubernetes into the CERN Batch Service, the impact its adoption has in daily operations, a comparison on resource usage efficiency and the experience so far evolving our infrastructure towards this model.

1 Introduction

The Batch Service, part of CERN's IT department, is responsible for providing Tier-0 compute power to the Worldwide LHC Computing Grid (WLCG). The principle goals are to process CPU and I/O intensive workloads, ensure fair-share among various user groups, and to maximize the utilisation, throughput and efficiency of the available resources.

Using HTCondor [1] as the job scheduling system the service currently offers more than 200K cores of compute power to 500 monthly unique users. These resources are provisioned in 20K virtual machines, belonging to more than 40 OpenStack projects, and located in multiple data centers. In addition to these resources hosted on-premises, extra capacity is

*e-mail: luis.fernandez.alvarez@cern.ch
**e-mail: olga.vladimirovna.datskova@cern.ch
***e-mail: ben.dylan.jones@cern.ch
****e-mail: gavin.mccance@cern.ch

spawned in external clouds. This forms a heterogeneous pool with different available configurations —hardware, operating systems, kernel versions, virtualisation technologies, physical location—.

This complex infrastructure is managed using a tool-set initially designed in 2012 [2]. The primary goal at the time was to redesign the toolkit used in the CERN Computer Centre, to benefit from open source technologies as well as adopting cloud technologies. The outcome of this effort was the Agile Infrastructure project; the ecosystem of tools and procedures that have been the basis of efficient IT operations ever since.

However, in the intervening years, the technological context has changed significantly. Container technologies, such as Docker [3], have revolutionised how the IT community thinks about many of its traditional established processes and workflows. A vibrant ecosystem of tools has emerged, with Kubernetes becoming de-facto standard for resource orchestration, and many other projects filling operational or functional gaps in this new IT paradigm. In the context of the IT department, the infrastructure functionality has grown during these years, from container orchestration engines with OpenStack Magnum [4, 5] to ()api for baremetal provisioning with OpenStack Ironic [6, 7]. Finally, the Batch Service has faced not only the challenge of scaling-up the pool but to integrate opportunistic resources [8] and to also explore the exploitation of external clouds to expand capacity [9, 10]. In a rapidly changing environment, some constants remain: the compute demand keeps growing, and the IT budget is still tight.

Given this new scenario it was decided to evaluate how the Batch Service could leverage the new technology context to better tackle its main challenges. On the one hand, given that the mandate of the service is to make an efficient usage of the resources, *would it be possible to optimise the compute capacity by provisioning baremetal nodes?* The current deployment based on virtual machines was driven by the functionality available in the cloud api back in 2012, but new baremetal cloud apis could bring extra performance without operational overheads. On the other hand, *could the service benefit from adopting Kubernetes to improve operations?* Leveraging Kubernetes built-in features could simplify homegrown tooling, as well as provide a deployment model more suitable for provisioning extra capacity in external clouds.

With the aim of answering these questions and exploring how such provisioning model would impact the service, a prototype cluster has been designed, deployed and tested. The details of such prototype are described in this article.

2 Prototype

In order to evaluate the new provisioning model —Kubernetes on baremetal— for compute capacity, a new HTCondor cluster was built, independent from the main production systems. The cluster comprises 100 Intel®Xeon®E5-2630 machines with Simultaneous Multi-Threading (SMT) enabled, providing a total of 3200 logical cores.

The machines, as shown in figure 1, are provisioned in two different ways: 50% of the capacity is provisioned as Puppet-managed virtual machines, while the other half is provisioned as a Kubernetes baremetal cluster. Given the goal of these tests to measure the impact on compute resources utilisation and operation, the control plane deployment —HTCondor schedds and central managers— is left untouched.

- Puppet managed nodes are provisioned using 8 core OpenStack Nova virtual machines, this flavor keeps the ratio of 4 machines per hypervisor that is used in the vast majority of the infrastructure. Among other configuration, Puppet installs and configures condor_startd, the daemon that connects to the HTCondor pool.

- Kubernetes clusters are created using OpenStack Magnum. The Kubernetes nodes take the full host and run condor_startd as a DaemonSet. In addition to this main components, HashiCorp Consul [11] is deployed to register all the nodes in this model, as well as the control plane.

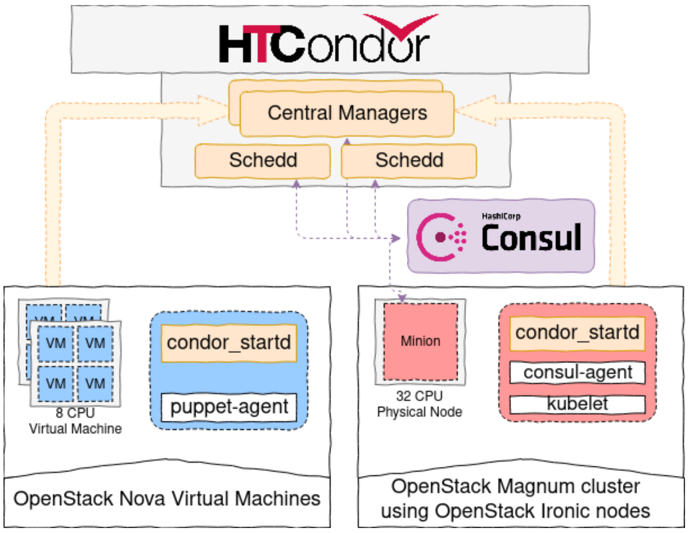

Figure 1. Overview of the resource configuration used in the prototype HTCondor pool

The proposed prototype has all the elements that a production cluster would have, allowing the evaluation of the impact on operations. Additionally, having a homogeneous hardware platform for all nodes simplifies the task of comparing benchmark results. Both operations and benchmark results are discussed in the following sections.

3 Operations

Operations is the main area impacted while transitioning to this model, not only because of the new tooling but because there is a paradigm shift on how the system is deployed. Changing the operational model should lead to efficiency gains, by reducing friction between components, increasing automation capabilities and reducing technical debt [12] in the systems. Operational tasks are defined by the life-cycle of resources, from provisioning and maintenance to decommissioning. In these sub-sections, the most important operations that are impacted by this new setup will be described.

3.1 Bootstrap

Bootstrapping the compute nodes in the current deployment based on Puppet, relies on homegrown tooling that plugs together the different components of the infrastructure: OpenStack, Foreman, Puppet and other CERN-specific services.

While evaluating Kubernetes, there was also an opportunity to explore the alternatives available to follow an Infrastructure as Code (IaC) approach in the deployment. The landscape of IaC tools is dominated by Terraform [13] and AWS CloudFormation [14], with new tools also gaining some traction recently, such as Pulumi [15]. Given previous experience at

CERN with Terraform, and the availability of providers for all required components, it was chosen as the tool to automate the new deployment.

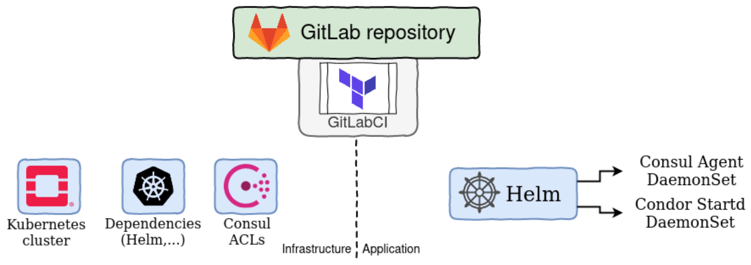

Figure 2. Terraform providers involved in cluster boostrapping

Figure 2 shows the main Terraform providers used during cluster bootstrapping. Terraform manifests are stored in GitLab and executed from GitLab Continuous Integration/Continuous Deployment (CI/CD) when there is a change to the manifests. It is responsible for interacting with OpenStack Magnum to create the Kubernetes cluster, adding the required dependencies in the newly created cluster, as well as creating the tokens for the cluster in order to be able to join Consul. Once the cluster is ready, Helm [16] is used to deploy the applications required: the Consul agent DaemonSet and the HTCondor startd DaemonSet.

This approach should introduce more flexibility while working on other clouds. On the one hand, it makes a clear distinction on what is used to provision the Kubernetes cluster — the only component that would end-up being cloud dependant—, and what is used to deploy the application —Helm charts, that should remain cloud independent—. On the other hand, having an infrastructure not depending on CERN provided services, such as Puppet masters or PuppetDB instances, ensures components and procedures are more adaptable to deploy services on other clouds and locations, improving the scalability of the service.

3.2 Node discovery and authentication

The second set of operations that was important to evaluate, is discovery and authentication of worker nodes in the infrastructure. In the current model the discovery is done by querying the central PuppetDB service, getting a fresh list of nodes on every Puppet run. In Kubernetes, it was decided to register all the nodes in Consul, registering a condor_startd services from every worker and making the control plane query Consul to populate the list of valid nodes using consul-template [17]. The choice of Consul is based on its high-level abstractions for service discovery, as well as functionality such as distributed locking that could help improve other areas of infrastructure automation. The result is a discovery mechanism that is updated immediately after a node joins the cluster, rather than waiting for a period of time, in the old system, before Puppet runs added the necessary config asynchronously. Another positive aspect of this approach is the Consul federation capabilities, having the possibility of spawning them in the public cloud to federate later with a CERN central instance. Further improvements to the prototype would require testing the right architecture to scale-out the Consul cluster.

Authentication is left untouched, relying on x509 certificates. This mechanism is still applicable as the CERN OpenStack Magnum instances are capable of issuing Grid certifications for the minions. In future iterations of this prototype other mechanisms such as token authentication [18] will be evaluated.

3.3 Draining

Through the life-cycle of compute nodes in the Batch Service, there is a dominant recurrent task: draining. There are many scenarios where draining of nodes is required, such as: security patching, kernel upgrades, hardware replacement and hardware decommissioning.

Draining events which require more coordination are hardware interventions, as they involve other teams and occur more often. Up to now, the communication of a hardware intervention would follow these steps:

1. The repair team responsible of the hardware at CERN data centres would identify an issue in the hypervisor. At this level, the monitoring tooling would be provided by the central monitoring service in the IT department.

2. Using a Rundeck [19], the repair team interacts with the Cloud team to disable the hypervisor in OpenStack and triggering a notification of the affected virtual machines. This is a planned intervention.

3. The Batch team listens to those notifications and marks the selected nodes as broken.

Running on baremetal implies that the Cloud layer is partially gone and the repair team should interact differently with the service managers in order to make sure that a machine is drained before they start their interventions. In such scenarios, the sequence of events would be as follows:

1. The repair team receives an alert from a baremetal minion in Kubernetes. This mechanism could be implemented using Collectd [20] as it is done at the moment or implementing new probes based on tools that are part of the Kubernetes ecosystem, for example a combination of Prometheus [21] and Alertmanager [22], as well as node-problem-detector [23].

2. The notification which in this case is used to trigger the draining could just rely on the built-in Kubernetes command "drain". This command starts the draining process for the given minion, sending a signal to the containers that can be captured by HTCondor to start draining the slots. It uses the parameter "terminationGracePeriodSeconds" defined by the application to set the maximum time it will wait before killing the applications.

Further automation can be implemented as well with other tools such as draino [24]. This tool is capable of draining nodes automatically based on the information provided by the node-problem-detector. In this case, no access to the cluster would be required for the repair team, and an implicit contract could be made such that they can perform the intervention one week after the alarm has been raised.

As stated above, the Cloud team is partially unaware of the new model. However, these default configurations could be provided as default settings in the cluster provisioning system such as OpenStack Magnum.

3.4 Upgrades

Keeping the nodes up to date is another important implicit operation while running such a system at large scale. At the moment, nodes are kept up to date based on a combination of updated base images [25], automated package upgrades and the Puppet configuration management system to provide new changes as they are required.

In the new model, container images can be maintained in a similar automated fashion. The Kubernetes DaemonSet deploying condor_startd uses an image containing all the dependencies provided in standard worker nodes. This approach has the impact on requiring a full draining of the jobs when a new image needs updating, leading to resource usage inefficiencies. However, it could be mitigated by reducing the condor_startd image to a minimal image, and wrap the jobs in containers automatically via the HTCondor Docker Universe. This approach would mean that jobs will pick-up always the latest image available without impacting on capacity. Only upgrades to HTCondor itself would require draining of the nodes as it happens today.

Additionally, for configuration changes, a combination of Helm and Consul could be used to provide dynamic configuration changes to the application. Tools like envconsul [26] can propagate automatically new changes to configuration files and trigger the configuration of the daemons running.

4 Benchmarking

Evaluating the impact on operations is critical, as it is directly related to how much time the team spends doing operations, and thus how efficient the staff dedicated to the service is optimised. On the other hand, a concern running a batch service is how efficiently the resources given are being used: when running at this scales getting a few percent more out of the machines can have a significant overall impact.

In the proposed prototype, the element bringing the potential improvement in efficiency is the baremetal provisioning of the Kubernetes minions. This transition should remove the virtualisation overhead, estimated by the Cloud team in the past to be around 3-4 percent [27], but there are other factors that could impact efficiency as many components in the provisioning stack change, thus it is important to define a systematic benchmark process to spot potential sources of inefficiency.

While defining benchmarking plans, it was decided to mimic as much as possible the workflow a normal user would follow and run the benchmarks via HTCondor, with the only simplification that there would be only a single client in the whole cluster running the same type of jobs. The job payload is created based on the effort from the Benchmark working group on their hep-workloads [28] project. The jobs are configured to run on 8 core slots, running 8 copies and 1 thread if the payload is single-threaded, or 1 copy and 8 threads if multi-threaded.

The cluster is configured to allow jobs to specify which subset of resources to target by the means of a boolean attribute named "+WantKubernetes" in its submission file. The execution of the benchmarks is wrapped in a script that takes care of reading the input from the arguments, launching the benchmarks and sending the output to a central ElasticSearch instance in the IT department. Apart from the default tags provided by the workloads, extra information is added to enrich (e.g: cluster id, job id, project and infra). The data can be then easily processed for visualization with tools like Grafana or exported for further analysis with tools like R.

4.1 Fine-tuning the deployment

The first results obtained showed some discrepancies with the initial expectations running on baremetal. Baremetal nodes showed very spread scores compare to virtual machines, and lower throughput. These initial results were obtained running Compact Muon Solenoid (CMS) digitization benchmarks and can be seen in table 1.

Table 1. Initial results for CMS digitization benchmark. The throughput is expressed as "mean ± standard error"

Type	Events	Job Throughput (events/s)
Virtual Machines	64000	$0.8178 \pm 3.7 \times 10^{-4}$
Baremetal	64000	$0.7294 \pm 8.3 \times 10^{-3}$

These results prompted a detailed review into the configuration of the baremetal deployment, to identify the cause of this loss performance degredation. Having both type of nodes on very same hardware helped link the issue with the configuration, rather than a platform specific performance issue.

This investigation led to the conclusion that the Non-Uniform Memory Access (NUMA) [29] topology was playing an important role in the way the kernel was running the benchmarks. As mentioned in the introduction, the virtual worker nodes were provisioned using standard 8 core flavours. The OpenStack flavors defined in projects assigned to the batch service are configured with extra meta-data to ensure the virtual machines are pinned to different NUMA nodes, as it was previously found that not doing so would incur a performance penalty [27]. Having the VMs pinned to a single NUMA node ensures that its data will be located in the memory node available in that NUMA node, without incurring in the penalty of accessing the memory located in a different NUMA node.

In order to apply the same principles to the baremetal deployment, and help the kernel scheduling jobs pinned to the same NUMA node, it was decided to spawn multiple HTCondor startd daemons and link them per NUMA node. The schema is shown in fig 3. Using HTCondor ability to spawn multiple daemons of the same type, the baremetal nodes have been split in two halves, then exposed in the cluster with the naming pattern "slot1@numa[0|1]@<hostname>".

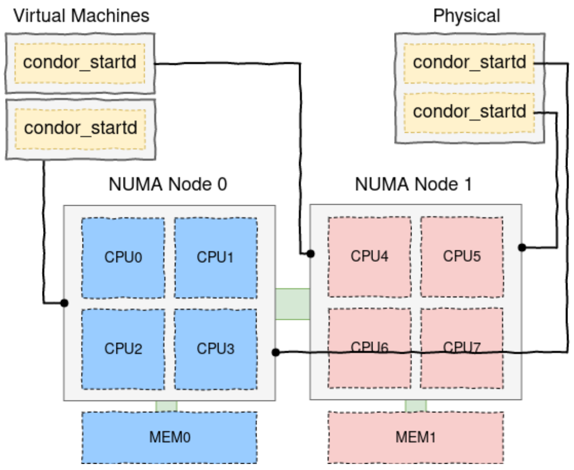

Figure 3. NUMA pinning is done at the virtualization layer on virtual nodes, and at the process daemon level on baremetal.

The results obtained after applying the aforementioned modifications are shown in the following section.

4.2 Results

After applying the modifications to pin the daemons to different NUMA nodes, the results started to be less spread and the results matched the initial expectations, i.e. that the performance on baremetal should be better.

Table 2. Job throughput (events/second) improvements running the hep-workloads on baremetal compared to virtual machines. The throughput is expressed as "mean ± standard error" (ST=Single-Thread, MT=Multi-Thread)

Benchmark	Type	Virtual Machine	Baremetal	Increase
ALICE GEN-SIM	ST	$0.1494 \pm 9.9 \times 10^{-2}$	$0.1531 \pm 2.1 \times 10^{-4}$	2.44%
ATLAS SIM	MT	$0.0120 \pm 5.9 \times 10^{-6}$	$0.0128 \pm 6.1 \times 10^{-6}$	5.98%
CMS GEN-SIM	ST	$0.1681 \pm 9.9 \times 10^{-5}$	$0.1777 \pm 1.2 \times 10^{-4}$	5.37%
CMS DIGI	MT	$0.8178 \pm 3.7 \times 10^{-4}$	$0.8562 \pm 2.7 \times 10^{-4}$	4.49%
CMS RECO	MT	$0.4635 \pm 1.4 \times 10^{-4}$	$0.4853 \pm 1.4 \times 10^{-4}$	4.49%
LHCb GEN-SIM	ST	$0.0176 \pm 7.0 \times 10^{-6}$	$0.0188 \pm 1.3 \times 10^{-5}$	6.38%
LHCb GEN-SIM (CVMFS)	ST	$0.0173 \pm 6.9 \times 10^{-6}$	$0.0189 \pm 7.3 \times 10^{-6}$	8.48%

The performance improvements obtained in the benchmarks that were executed is shown in Table 2. The results show improvements over the 4% attributed to the virtualisation overhead. In one particular case —LHCb GEN-SIM— it was possible to run the benchmark without relying on the image with cached data and access directly CernVM File System (CVMFS) to get the required data. In this particular case, the gain reached 9%. Due to the scope of this prototype, there was no time to explore this scenario more in detail, but it can be potentially connected to the impact on having multiple CVMFS caches when running on virtual machines —remember that on a 32 CPU node it is usual to run 4 machines of 8 cores, leading to 4 CVMFS caches to be filled—.

5 Conclusions

Exploring the adoption of Kubernetes on the service by deploying a prototype similar to the production cluster has helped to focus on existing procedures, to have a better understanding of what it would entail when migrating the infrastructure to such model. From an operations point of view, there is now a better picture of the ecosystem of tools around Kubernetes that could help tackle some of the main procedures in the life-cycle of the infrastructure. Furthermore, it has been possible to identify the dependencies within the organisation that would require adaption of dependent procedures to conform with this new setup. At the performance level, the results obtained in the benchmarking of the prototype show that moving towards a baremetal service would gives a potential boost of performance in the order of 4% and 9%, for an amount that could satisfy the compute demands in the next LHC run.

Taking into consideration all these points, this new model — baremetal Kubernetes clusters — for provisioning resources has proved to be operationally feasible and could help to tackle some of the new challenges the service will be facing. The next steps to take in order to validate these initial conclusions are: to deploy a similar setup within the production cluster and route part of the WLCG jobs to confirm the results obtained, as well as keep improving the deployment model.

Moving towards such a model can be challenging, in particular in big IT organisation like CERN where many dependent units work together and adapting existing workflows and procedures might have a higher impact due to existing expertise or manpower available.

References

[1] C. for High Throughput Computing at UW-Madison, *Computing with HTCondor*, accessed February 12, 2020, https://research.cs.wisc.edu/htcondor/

[2] P. Andrade, T. Bell, J. van Eldik, G. McCance, B. Panzer-Steindel, M.C. dos Santos, S. Traylen, , U. Schwickerath, *Review of CERN Data Centre Infrastructure* (IOP Publishing, 2012), Vol. 396, p. 042002, https://doi.org/10.1088%2F1742-6596%2F396%2F4%2F042002

[3] D. Inc., *Docker*, accessed February 12, 2020, https://www.docker.com/

[4] OpenStack Community, *OpenStack Docs: Welcome to Magnum's Developer Documentation!*, accessed June 09, 2020, https://docs.openstack.org/magnum/latest/

[5] B. Noel, D. Michelino, M. Velten, R. Rocha, S. Trigazis, *Integrating Containers in the CERN Private Cloud* (IOP Publishing, 2017), Vol. 898, p. 092045, https://doi.org/10.1088%2F1742-6596%2F898%2F9%2F092045

[6] OpenStack Community, *OpenStack Docs: Welcome to Ironic's documentation!*, accessed June 09, 2020, https://docs.openstack.org/ironic/latest/

[7] S. Trigazis, A. Wiebalck, D. Abad, M. Kowalski, *Integrating ironic into cern's private cloud service* (2018), https://indico.cern.ch/event/676324/contributions/2981729

[8] Smith, David, Di Girolamo, Alessandro, Glushkov, Ivan, Jones, Ben, Kiryanov, Andrey, Lamanna, Massimo, Mascetti, Luca, McCance, Gavin, Rousseau, Herve, Schovancová, Jaroslava et al., *Sharing server nodes for storage and computer* (2019), Vol. 214, p. 08025, https://doi.org/10.1051/epjconf/201921408025

[9] B. Jones, G. McCance, C. Cordeiro, D. Giordano, S. Traylen, D.M. García, *Future Approach to tier-0 extension* (IOP Publishing, 2017), Vol. 898, p. 082040, https://doi.org/10.1088%2F1742-6596%2F898%2F8%2F082040

[10] C. Cordeiro, L. Field, B.G. Bear, D. Giordano, B. Jones, O. Keeble, A. Manzi, E. Martelli, G. McCance, D. Moreno-García et al., *CERN Computing in Commercial Clouds* (IOP Publishing, 2017), Vol. 898, p. 082030, https://doi.org/10.1088%2F1742-6596%2F898%2F8%2F082030

[11] HashiCorp, *Consul by HashiCorp*, accessed June 09, 2020, https://www.consul.io/

[12] Wikipedia contributors, *Technical debt — Wikipedia, the free encyclopedia* (2020), accessed March 6, 2020, https://en.wikipedia.org/w/index.php?title=Technical_debt&oldid=938845015

[13] HashiCorp, *Terraform*, accessed February 12, 2020, https://www.terraform.io

[14] Amazon Web Services, Inc., *AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning*, accessed February 12, 2020, https://aws.amazon.com/cloudformation/

[15] Pulumi, *Pulumi - Modern Infrastructure as Code*, accessed February 12, 2020, https://www.pulumi.com/

[16] Helm Authors, *Helm, The package manager for Kubernetes*, accessed June 09, 2020, https://helm.sh/

[17] HashiCorp, *Template rendering, notifier, and supervisor for hashicorp Consul and Vault data*, accessed February 12, 2020, https://github.com/hashicorp/consul-template

[18] C. for High Throughput Computing at UW-Madison, *Token Authentication*, accessed February 12, 2020, https://htcondor.readthedocs.io/en/latest/admin-manual/security.html#token-authentication

[19] Rundeck, *Rundeck | Runbook Automation*, accessed June 14, 2020, `https://www.rundeck.com/`

[20] Florian Forster, *collectd – The system statistics collection daemon*, accessed June 14, 2020, `https://collectd.org/`

[21] Prometheus Authors, *Prometheus - Monitoring system & time series database*, accessed February 12, 2020, `https://prometheus.io`

[22] Prometheus Authors, *Alertmanager*, accessed February 12, 2020, `https://prometheus.io/docs/alerting/alertmanager/`

[23] *node-problem-detector*, accessed February 12, 2020, `https://github.com/kubernetes/node-problem-detector`

[24] Planet Labs, *draino: automatically cordon and drain Kubernetes nodes based on node conditions*, accessed February 12, 2020, `https://github.com/planetlabs/draino`

[25] J. Schovancova, A. Di Girolamo, A. Fkiaras, V. Mancinelli, *Evolution of HammerCloud to commission CERN Compute resources* (2018), Vol. 214, p. 03033. 7 p, `https://cds.cern.ch/record/2646247`

[26] HashiCorp, *Launch subprocesses with dynamically populated environment variables read from Vault and Consul.*, accessed February 12, 2020, `https://github.com/hashicorp/envconsul`

[27] A. Wiebalck, S. Crosby, T. Bell, *Optimisations of the compute resources in the cern cloud service* (2015), `https://indico.cern.ch/event/384358/contributions/909247/`

[28] HEPiX Benchmarking Working Group contributors, *Collection of HEP workloads for benchmarking purposes.*, accessed February 10, 2020, `https://gitlab.cern.ch/hep-benchmarks/hep-workloads/`

[29] The Linux kernel development community, *What is NUMA?.*, accessed February 25, 2020, `https://www.kernel.org/doc/html/v4.18/vm/numa.html`